Comparison and Evaluation of Web Technologies in PHP and Node.js

***Abstract:***

In the modern world, web applications play an important role for many industries. Large scale and high concurrency are important factors for the new era of web applications. The Web technologies involved in developing these applications play an important role. Many technologies such as PHP, ASP.NET, Node, Python, Ruby etc are available in the market for companies to achieve this goal. PHP has been in the web development industry for many years and a large number of web applications still run on PHP while Node has gained popularity in the last few years. Node has become a popular technology to build data-intensive web applications. To study and analyse the performance of Node and PHP, we use benchmark and scenario tests. The results obtained have some valuable performance data, showing that PHP handles much less requests than Node in a certain time. Results show that Node is lightweight and efficient, which is an idea fit for I/O intensive websites between the two, while PHP is relatively more suitable for small and middle scale applications.

Introduction

An application server is a framework that provides developers with facilities to create web applications and a server on which we can run them. Application Server Frameworks contain a detailed service layer model where the application server acts as a group of components accessible to the code developer through a regular API outlined for the platform itself. For web applications, these parts are sometimes performed within the same running environment as their web server(s), and their main job is to support the development of dynamic pages. However, several application servers target way more than simply web page generation: they implement services like clustering, fail-over, and load-balancing, thus developers can concentrate on implementing the business logic.

Application server also refer to the computer hardware on which the services run.

Application servers are platforms where web applications or desktop applications run. Application servers comprise of web server connectors, PC programming languages, runtime libraries, database connectors, and the organization code expected to be deployed, design, oversee, and connect these parts on a web host. An application server keeps running behind a web server (e.g. Apache or Microsoft Internet Information Services (IIS)) and quite often before a SQL database (e.g. PostgreSQL, MySQL, or Oracle). Web applications are codes which keep running on application servers and are composed in the language(s) the application server supports and call the runtime libraries and parts the application server offers.

Numerous application servers exist. The decision of choosing a server architecture impacts the cost, execution, reliability, adaptability, and viability of a web application. Exclusive application servers give system benefits in an all-around characterized yet proprietary manner. The application engineers create programs as per the specification of the application server. This project aims to compare the performance of a selected set of web server technologies and analyse the results based on various circumstances and conclude with which web server architecture should be used under what situations. Along with comparing the various web technologies, this project also aims to test t The benchmarking of these various web technologies will be done using two popular benchmarking tools called ‘LoadRunner’ and ‘JMeter’. The project also aims to compare the

.NET Framework

Microsoft positions their middle-tier applications and services infrastructure in the Windows Server operating system and the .NET Framework technologies in the role of an application server.[5] The Windows Application Server role includes Internet Information Services (IIS) to provide web server support, the .NET Framework to provide application support, ASP.NET to provide server side scripting, COM+ for application component communication, Message Queuing for multithreaded processing, and the Windows Communication Foundation (WCF) for application communication.[6]

PHP app servers

PHP application servers are used for running and managing PHP applications.

Zend Server, built by Zend Technologies, provides application server functionality for the PHP-based applications.

appserver.io, built by TechDivision GmbH is a multithreaded application server for PHP written in PHP.

PHP

.NET

In the rapid development of Web today, many sites are faced with new problems, such as the problem of multiuser requests and high concurrency. The dynamic scripting language JavaScript has become enormously popular for client and is widely used in Web development. Node stands for one new technology in JavaScript. Node is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications [1]. Node.js uses an event-driven, non- blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices [1]. Node.js popularity surveys performed by official website indicate that the average downloads are over 35,000 since the version 0.10 released in March 2013. Corporations are quickly realizing the importance of Node.js and five major PAAS providers have supported Node.js [2]. Nowadays, JavaScript has been the first popular language in GitHub with 177,352 repositories and growing [3]. And talking about evaluation of Web technologies’ performance, many researchers have done the related work. But the work described in this thesis differs from others in two aspects. Firstly, we consider from both objective systematic tests (benchmark) and realistic user behaviour tests (scenario).

Not long ago even a 2-second page response time was considered as an acceptable one. However, web users have become increasingly impatient when it comes to speed these days. Earlier, speed was considered a feature and now it is deemed a necessity. Additionally, technological innovation in mobile space has raised the bar for speed. Hence, speed makes a lot of economic sense now. A recent research found that 250-450 milliseconds are the magical numbers that decide the winner in the race of web speed []. Research also indicates that the slower the site, the lesser would be the number of clicks and transactions performed on the site which would eventually result in the loss of users.

In order to perform these tests on the various Web technologies we will make use of the testing tools such as JMeter and LoadRunner. In addition to testing the various Web technologies, this project will also focus on the quality of the testing tools, the differences between them and if they produce different results under certain circumstances. Complex systems make increasing demands on web servers, multiple objects can interfere when one process is handling a request for a specific user. High volumes can overwhelm systems if they are not scaled correctly. Fixes need to be identified early in the project so that server crashes and vulnerabilities can be caught in advance Clients have scalability concerns and we must warranty some level of scalability with industry accepted metrics. In order to achieve this, our web servers

# Purpose of the project

This paper focuses on the impact on Web performance from two different Web technologies: Node and PHP. The security and scalability issues are beyond the scope of the thesis. We mainly use the benchmark tests and scenario tests. In addition, one universal method of Web development technique’s evaluation based on the performance comparison is proposed in the paper, which can be used to evaluate any new Web technology. The main contributions of this paper are listed as follows.

(1) We consider new web technology Node in our experiment and analyse the results of it. Then we compare it with PHP making a conclusion of which situation they ought to be used.

(2) By means of benchmark tests and scenario tests, we can evaluate performance from both objective systematic tests (benchmark) and realistic user behaviour tests (scenario). There is often a dual impact on Web server performance, from the calculation, and from the number of users. The research herein has taken each of these effects in account.

The rest of this research is organized as follows - Section 2 discusses related work. Section 3 describes the test bed and configurations in the research. Section 4 details the methodology and experimental design of tests. Section 5 presents and analyses the results of all tests. Section 6 makes a conclusion of the paper with a summary of study and a future direction.

Node

The official website (http://www.nodejs.org) defines Node as “a platform built on Chrome’s JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices [1].”

Regardless, JavaScript is the world's most famous programming languages. If you have done any programming for the web, it's unavoidable. JavaScript, in view of the sheer reach of the web, has satisfied the "compose once, run anywhere" dream that Java had back in the 1990s.

Around the season of the Ajax insurgency in 2005, JavaScript went from being a "toy" language to something individuals wrote genuine and noteworthy projects with. A portion of the eminent firsts were Google Maps and Gmail, yet today there are a large group of web applications from Twitter to Facebook to GitHub.

JavaScript has for some time been the true standard for frontend side web development. While about all frontend code is composed in JavaScript, server-side development is a variety of choices between PHP, Java, and various different technologies. Life as a web engineer would be substantially more straightforward if a single language was utilized all around. Since JavaScript overwhelms in the browser, it bodes well to utilize it on the server too.

The idea of server-side JavaScript is not a new one. Netscape initially introduced JavaScript into the server world in 1994. Since that time, a lot of projects have endeavoured, and failed, to advance JavaScript as a server-side language. Execution, or scarcity thereof, restricted JavaScript from picking up a genuine a dependable balance in the server space.

Throughout the years, JavaScript has seen gigantic upgrades in performance. Because of its pertinence in the program, enormous players like Google have contributed a considerable measure of time and cash to make JavaScript as fast as possible. In 2009, Ryan Dahl of Joyent, put the large part of that recently discovered execution to great use on the server when he made the Node.js structure. Dahl assembled Node.js over Google's V8 JavaScript engine. V8 is a similar engine that has given Google Chrome its astounding JavaScript performance, and helped it turn into the most well-known browser on the planet.

# Technical details

**Threading**

Node.js works in a single threaded environment, utilizing non-blocking I/O calls, enabling it to support a huge number of simultaneous connections without incurring the cost of thread context switching [17]. The idea of sharing a single thread among every one of the requests that utilization the observer design is for building exceptionally concurrent applications, where any function performing I/O must utilize a callback. With a specific end goal to accommodate the single-threaded event loop, Node.js uses the libuv library that, utilizes a fixed-sized thread pool that is in charge of a portion of the non-blocking async I/O operations [18].

A drawback of this single-threaded approach is that Node.js does not permit vertical scaling by increasing the amount of CPU cores of the machine it's running on while not using a further module, like cluster, StrongLoop process Manager, or pm2. However, developers can increase the default range of threads within the libuv thread pool; these threads are probably to be distributed across multiple cores by the server software system [19].

Execution of concurrent tasks in Node.js is handled by a thread pool. the main thread decision functions post tasks to the shared task queue that threads within the thread pool pull and execute. Inherently non-blocking system functions like networking interprets to kernel-side non-blocking sockets, whereas inherently blocking system functions like file I/O run in an exceedingly block method on its own thread When a thread in the thread pool completes a task, it informs the main thread of this, which wakes up and execute the callback. As callbacks are handled synchronously on the main thread, long lasting computations and other CPU-bound tasks will freeze the whole event-loop till completion.

**V8**

V8 is the JavaScript execution engine built for Google Chrome and open-sourced by Google in 2008. Written in C++, V8 compiles JavaScript source code to native machine code instead of interpreting it in real time [18].

Node.js makes use of libuv to handle asynchronous events. Libuv is an abstraction layer file system and network functionality on each Windows and POSIX-based systems like UNIX operating system, macOS, OSS on NonStop, and Unix.

The core functionality of Node.js resides in a JavaScript library. The Node.js bindings, written in C++, connect these technologies to each other and to the operating system.

**Package management**

npm is an in-built package manager for the Node.js servers. It is utilized to install Node.js programs from the npm registry, sorting out the installation and administration of third-party Node.js programs. npm isn't to be mistaken for the CommonJS ‘require()’ statement. It is not utilized to load code; rather, it is utilized to install code and manage dependencies from the command line. The bundles found in the npm registry can extend from basic helper libraries, for example, Lodash to task runners suck as Gulp.

**Unified API**

Node.js may be combined with a browser, a database supporting JSON information (such as Postgres, MongoDB, or CouchDB) and JSON for a unified JavaScript development stack. With the difference of what were basically server-side development patterns like MVC, MVP, MVVM, etc., Node.js permits the reuse of the same model and service interface between client-side and server-side.

**Event loop**

Node.js registers itself with the software system so as to be notified once a connection is created, and also the OS can issue a callback. inside the Node.js runtime, every connection could be a small heap allocation. historically, comparatively heavyweight OS processes or threads handled every connection. Node.js uses an event loop for scalability, rather than processes or threads [74]. In distinction to different event-driven servers, Node.js's event loop doesn't need to be known as explicitly. Instead callbacks are outlined, and also the server mechanically enters the event loop at the end of the callback definition. Node.js exits the event loop once there are not any any callbacks to be performed.

# Features of Node

Some of the remarkable features that make Node.js a first choice for software architects are listed below:

**Asynchronous and Event Driven** − All APIs of Node.js library are non-concurrent, that is, non-blocking. It fundamentally suggests that a Node.js based server never waits for an API to return information. The server moves to the accompanying API ensuing to calling it and a notice system of Events of Node.js encourages the server to get a response from the past API call.

**Very Fast** – Since Node is built on on Google Chrome's V8 JavaScript Engine, it is extremely fast in at executing code.

**Single Threaded but Highly Scalable** − Node.js utilizes a single threaded model with event callbacks. Events encourage the server to react in a non-blocking way and makes the server scalable compared to traditional servers which give restricted access to threads to deal with requests. Node utilizes a single threaded program to provide services to a substantially bigger number of requests than traditional servers like Apache HTTP Server.

**No buffering of data** – Data is never buffered in Node. The applications output the data in chunks.

# Node Concepts

Node.js is a relatively new environment based on the Chrome V8 JavaScript engine that enables you to run JavaScript server-side. Its ubiquity is expected to some degree to its ability to enable you to compose what is regularly referred to as "fullstack JavaScript" — JavaScript on both the client and the server.

**Non-blocking**

The question of of whether an operation is blocking or non-blocking refers to the fact that it must finish before the next operation begins. Non-blocking operations are said to be asynchronous and blocking operations are said to be synchronous. Node is non-blocking — i.e. operations don't have to happen consecutively.

**Timers**

Node contains a timers module that functions similar to the window.setTimeout() native JavaScript method. One noteworthy distinction, nonetheless, is that there are also other comparative methods, for example, setInterval() and setImmediate(), which are for functions that should run an unending number of times (looping functions), and executes code toward the end of the present event loops cycle.

**The Event Loop**

The event loop is the thing that enables Node to be non-blocking. More or less, this implies there is one primary loop tuning in for events. At the point when an event is discovered, the main loop dispatches a callback function.

**Memory Management**

# Since Node is single-threaded, that implies that every one of your users will be sharing the same memory allocation. At the end of the day, unlike to in the browser, you must be mindful so as not to store user-specific information in closures where different connections can affect it.

# Middleware

**Express** is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. Express provides a thin layer of fundamental web application features, without obscuring Node.js features [17].

Express is the most prominent framework for Node applications, and it highlights middleware utilizing continuation passing. When you need to run a similar code for conceivably a wide range of routes, the perfect place for that code is likely middleware.

Middleware is a function that gets passed the request and response objects, alongside a continuation function to call, called next(). Envision that you need to add a requestId to each request/response pair with the goal that you can follow them back to the individual request when you're troubleshooting or debugging your logs for something.

You can write some middleware like this:

require('dotenv').config();

const express = require('express');

const cuid = require('cuid');

const app = express();

// request id middleware

const requestId = (req, res, next) => {

const requestId = cuid();

req.id = requestId;

res.id = requestId;

// pass continuation to next middleware

next();

};

app.use(requestId);

app.get('/', (req, res) => {

res.send('\n\nHello, world!\n\n');

});

module.exports = app;

PHP

PHP (recursive acronym for PHP: Hypertext Preprocessor) is a widely-used open source general-purpose scripting language that is especially suited for web development and can be embedded into HTML [18].

Instead of lots of commands to output HTML (as seen in C or Perl), PHP pages contain HTML with embedded code that does "something" (in this case, output "Hi, I'm a PHP script!"). The PHP code is enclosed in special start and end processing instructions <?php and ?> that allow you to jump into and out of "PHP mode."

What distinguishes PHP from something like client-side JavaScript is that the code is executed on the server, generating HTML which is then sent to the client. The client would receive the results of running that script, but would not know what the underlying code was. You can even configure your web server to process all your HTML files with PHP, and then there's really no way that users can tell what you have up your sleeve [18].

PHP began as a small open source venture that advanced as an ever-increasing number of people discovered how valuable it was. Rasmus Lerdorf released the primary rendition of PHP back in 1994.

* PHP is a recursive acronym for "PHP: Hypertext Preprocessor".
* PHP is a server-side scripting language that is embedded with HTML. It is utilized to manage dynamic content, session tracking, databases even build whole web based e-commerce websites.
* It is coordinated with various well-known databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.
* PHP is pleasingly fast in its execution, particularly when compiled as an Apache module on the Unix side. The MySQL server, once running, executes even extremely complex queries with tremendous results returned in record-setting time.
* PHP supports countless protocols, for example, POP3, IMAP, and LDAP. PHP included support for Java and distributed object architectures (COM and CORBA), making n-level improvement a plausibility for the first time.
* PHP Syntax is C-Like.

# What can PHP do?

PHP is essentially centred around server-side scripting, so you can do anything some other CGI program can do, for example, gather form information, produce dynamic page content, or send and get cookies. In any case, PHP can do significantly more.

There are three main areas where PHP scripts are used -

**Server-side scripting** - This is the most traditional and fundamental target field for PHP. In order to make this work you require three things: the PHP parser (CGI or server module), a web server and a web browser. You have to run the web server, with an associated PHP installation [19]. You can get to the PHP program output with a web browser, seeing the PHP page through the server. All these can keep running on your home machine if that you are simply experimenting with PHP.

**Command line scripting**. You can make a PHP script to run with no server or program. You just need the PHP parser to utilize it in the appropriate way. This kind of use is perfect for scripts consistently executed utilizing cron (on \*nix or Linux) or Task Scheduler (on Windows). These scripts can likewise be used for straightforward script processing tasks.

**Writing desktop applications.** PHP is presumably not the absolute best language to make a desktop application with a graphical UI, yet if you know PHP exceptionally well, and might want to utilize some advanced PHP includes in your client-side applications you can utilize PHP-GTK to compose such programs. Additionally, you can build cross-platform applications along these lines. PHP-GTK is an expansion to PHP, not accessible in the fundamental distribution.

PHP can be used on all major operating systems, including Microsoft Windows, Mac OS X, Linux, many Unix variants, RISC OS, and probably others. PHP has support for almost all of the web servers today. This includes IIS, Apache, and many others. And this consists of any internet server which could make use of the FastCGI PHP binary, like lighttpd and nginx. PHP works as both a module, or as a CGI processor.

So, with PHP, you have the liberty of selecting an operating system and a web server. Furthermore, you also have the choice of using procedural programming or object-orientated programming (OOP), or a combination of them both.

With PHP, you aren't confined to output HTML. PHP’s power includes outputting pictures, PDF documents or even Flash movies (with the usage of libswf and Ming) generated at the fly. You can additionally output easily any text, together with XHTML and another XML document. PHP can autogenerate these documents, and save them within the system, as opposed to printing it out, forming a server-side cache to your dynamic content material.

One of the strongest features in PHP is its support for a huge range of databases. Writing a database-enabled application is pretty easy with the help of one of the database specific languages (e.g. mysql), or using an abstraction layer like PDO, or connect to any database that supports the Open Database Connection standard popular via the ODBC extension. Other databases may additionally utilize cURL or sockets, like CouchDB.

PHP also has support for communicating with other services using protocols inclusive of IMAP, HTTP, LDAP, POP3, SNMP, NNTP, COM (on windows) and endless others. You may also open raw network connections and interact using every other protocol. PHP has aid for the WDDX complex information exchange among truly all web programming languages. When it comes to interconnection, PHP has support for instantiation of Java objects and the use of them transparently as PHP objects.

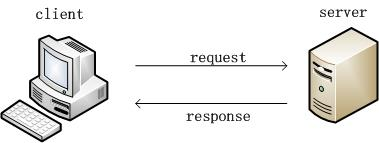
PHP has a powerful text processing feature, which incorporates the Perl compatible regular expressions (PCRE), and numerous expansions and functions to parse and get XML records. PHP standardises a large part of the XML augmentations on the strong base of libxml2, and expands the list of capabilities by including SimpleXML, XMLReader and XMLWriter support.

Implementation

# Experimental Environment

1. **Hardware Environment**

We set up a testbed consisting of two machines connected by 100-Mbps Ethernet, one for the client, the other for the server, including the Web server and Database server, as shown in Fig.1. We deployed Web server and Database server in one machine in order to avoid the impact of bandwidth during the test.



The server machine in our test runs Ubuntu Linux 13.04, with an Intel i3 3.30GHz processor, 4 GB of RAM and a 500 GB disk. The client machine runs Windows7 64-bit system, with an Intel i3 3.30GHz processor, 4 GB of RAM and a 500 GB disk. Both machines are connected by 100-Mbps Ethernet. In addition, all non-essential processes on the machines were disabled to minimize the consumption of resources and kept fair in our test.

1. **Server Configuration**

It’s mainly to compare Web development technologies of PHP, Python-Web and Node.js in our study. So, we choose several different modules to build the three environments as follows. The database uses Mysql5.5.

* Apache - Apache 2.4.9. It’s the newest stable version. Apache 2.4 uses a hybrid thread and process model in an attempt to improve the server’s performance, and it’s more scalable compared to Apache 1.3.
* PHP - PHP 5.5.12. The stable version that recently released. PHP is a popular general-purpose scripting language that is especially suited to Web development [13]. PHP is mainly used in dynamic Web page, including CLI (command line interface) and GUI (graphical user interface) program. It has the feature of good across-platforms and easy transplant.
* Python - Python 2.7. Python is a programming language that lets you work quickly and integrate systems more effectively [14]. It’s widely used in processing system administration tasks and Web programming since it was born in the 1990s. Its grammar is readable and clear. With the development of several years, Python has its own frames and easy to develop. In our test, we choose “WebPy” frame to make scenario tests.
* Node.js - Node.js 0.10. Node.js is a platform built on Chrome's JavaScript runtime for easily building last, scalable network applications. Node.js uses an event- driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real- time applications that run across distributed devices [1]. We choose “Express” frame to make tests in our experiment.

1. **Testing tools**

We divided the tests into benchmark tests and scenario tests. We use two testing tools in our experiment in order to make results more reliable. As to benchmark tests, we use ApacheBench (Ab), a tool in Apache. Ab can make requests in local Web server to ensure that the time is just processing time, not including data transmission time on the Internet or calculation time in local machine. Then we choose LoadRunner to make load tests to simulate the behaviour of users.

* ApacheBench - The stress testing tool in Apache 2.4. The nature of this kind test is based on Http, and it is a black-box of the Web server performance testing.
* LoadRunner - LoadRunner 11.00. It’s also a black-box testing tool. It can record browser behaviours, and simulate real situations to evaluate the performance of Web server.

# Testing Methodology

The experiment evaluated the results from two respects, one from the server to do benchmark tests, the other from the client to simulate the behaviour of users to do scenario tests. In all the tests, we must follow one-factor-at-a-time experimental design [16] to ensure the accuracy and effectiveness of tests.

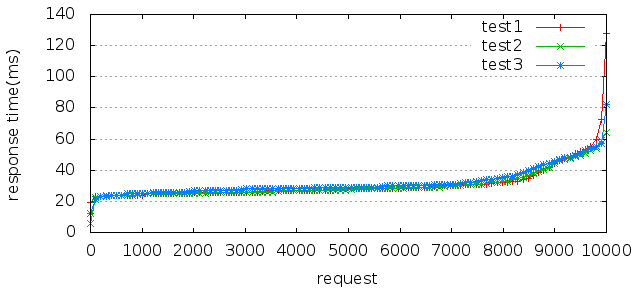
1. **Benchmark Test**
   1. **Benchmark Test Methodology**

According to one-factor-at-a-time experimental design, we make three fundamental tests – “Hello World”, “Calculate Value of Fibonacci” and “Select Operation of DB”. “Hello World” module is a basic module to build a good Web server, then output “hello word” and distinguish the differences of those three technologies. “Calculate Value of Fibonacci” module is to calculate some value of Fibonacci and evaluate the performance under compute-intensive tests. “Select Operation of DB” module is to compare different performance through querying some value of DB in the IO-intensive situation. Under all benchmark tests, we keep requests 10000, and then we change users from 10 to 1000. TABLE 1 summarizes the factors in our experiments.

|  |  |
| --- | --- |
| Requests | 10000 |
| Users | 10, 100, 200, 500, 1000 |
| Benchmark test module | Hello World, Fibonacci, DB Operation |
| Web development Methodology | Node, PHP, Python |

* 1. **Benchmark Test Configuration**

In the process of test, we found results of same module are similar. For example, we choose PHP to make three tests under requests 10000 and users 100. With the number of requests from 0 to10000, the results of three tests are as shown in Fig.2. The response time doesn’t have much difference in three tests with the increase of concurrency requests. The average time of three tests is separate 0.311ms, 0.305ms, and 0.319ms. So, we use one test result to evaluate performance in our experiment. In addition, we reboot server in every test to make sure fairness in whole experiment.



1. **Scenario Test**
2. **Scenario Test Methodology**

Scenario test aims to simulate realistic user behaviour. In our experiment, we divide scenario tests into two parts, one is “Login” scenario and the other is “Encryption” scenario. In terms of “Login” scenario, it mainly simulates concurrent users to login at the same time, and then compares the performance of three Web technologies in the real IO-intensive scenario. In the test, we choose 500 users as rendezvous because it appears some errors when users increase to 500. That also means the stress is beyond the maximum range Web server can stand. We then use correct results when users are 500. In the test, we make statistics of throughput, the average transaction response time, and “hits per second” to compare performance of those three technologies. Throughput displays the amount of data in bytes the Vusers receive from the server at any given second. “Hits per second” displays the number of hits made on the Web server by Vusers during each second of the load test. “Encryption” scenario is to simulate a process to encrypt users’ login password when users login. It’s mainly to compare performance in the simple real compute-intensive scenario. We choose the same rendezvous as “Login” scenario.

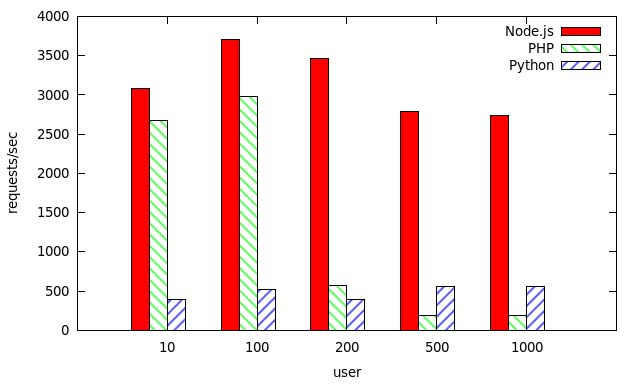
1. **Scenario Test Configuration**

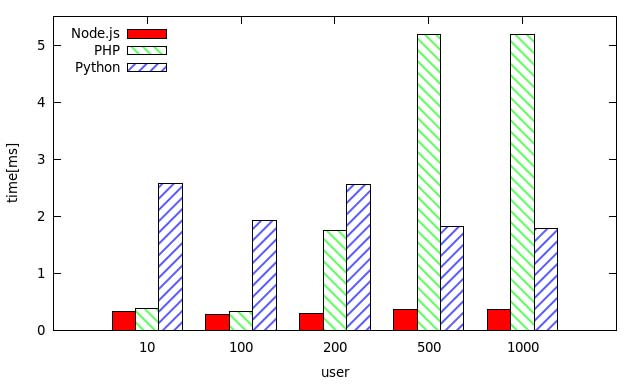
In order to make enough fair in our experiment, we make same configuration as benchmark tests. We reboot server in every scenario test.

# Experimental Analysis

1. **Results and Analyses of Benchmark Tests**
2. **„Hello World” module**

With growth of users, the performance of two technologies shows a trend of increasing before decreasing when keeping the requests at 10000. As FIG.3 and FIG.4 are shown, the “mean requests per second” is the highest which increases to 3703.5 times per second when the users of Node.js are 100. Meanwhile, the “mean time per request” is the shortest which is 0.27ms. Then the “mean requests per second” slows down and maintains a steady state around 2700. As to Python-Web, the “mean requests per second” keeps stable around 500 and the highest is 559.42. At this moment, the “mean time per request” is the shortest which is 1.788ms. To PHP, it is also at a peak when users are 100. The “mean requests per second” is 2977.54 at that time. The “mean time per request” is the shortest 0.336ms. With users increasing, the “mean requests per second” decreases to 200 and remains stable.

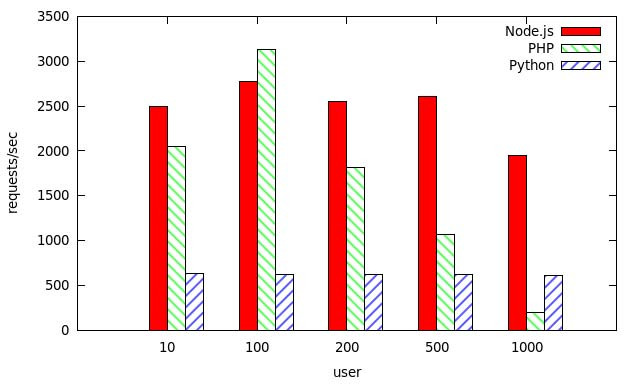


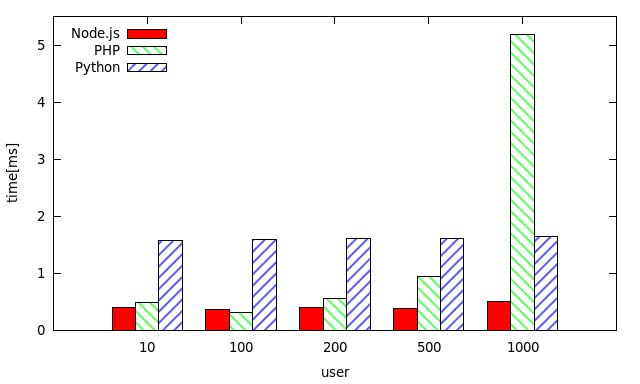


In short, the performance of Node.js is better than two others at the same number of users. The performance of Node.js is two times larger than PHP on the basic performance comparison and six to seven times larger than Python-Web. In addition, the current users that Node.js can hold are far more than PHP, let alone Python-Web. So, its performance is much better than PHP and Python-Web when there are lots of users.

1. **“Calculate Fibonacci” module**

FIG.5 and FIG.6 show the Web performance to calculate the tenth value of Fibonacci when requests are 10000 and users’ number increases on and on. In terms of Node.js, the “mean requests per second” is the highest value reaching up to 2777.72 times per second and the “mean time per request” is 0.36ms when users are 100. In addition, the “mean requests per second” of Node.js keeps from 2000 to 2800. The peak requests per second decreases 1.5 times comparing with the same condition of “Hello World” benchmark tests. But to Python-Web, this module has similar results as the last module, even better results, at the same condition. PHP is also increased to peak value when users’ number is up to 100, the “mean requests per second” at 3127.98. There is not much difference between it and “Hello World” module.



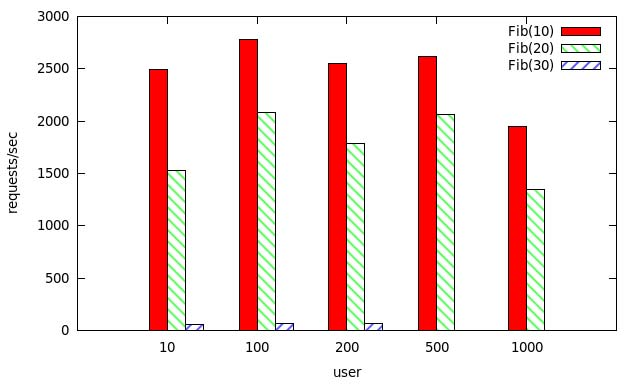


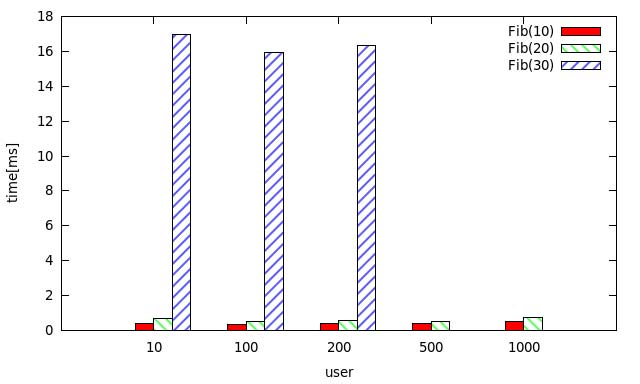
Obviously, the performance of Node.js is better than two others at “Calculate Fibonacci (10)” tests. But we found they all are very alike to “Hello World” module in the same condition, so we choose other values of Fibonacci to validate the results. We calculate the twentieth and thirtieth value of Fibonacci when users are 10. TABLE 2 shows the results of tests on above. The performance of three Web technologies decreases on different degree with the increase of Fibonacci. Considering the “Calculate Fibonacci (30)” test, all the tests decrease much in performance, especially PHP which “mean requests per second” drops from 2000 to 2. Besides PHP, Python-Web reduces from 600 to 3. Node.js also decreases much from 2500 to 60. This phenomenon means those three Web technologies all don’t adapt to compute-intensive application. However, Node.js performs better among the three in that test.

|  |  |  |  |
| --- | --- | --- | --- |
| **Web Technology** | **Calculate Fibonacci** | **Mean req/sec** | **Mean time/req** |
| Node | Fib(10/20/30) | 2491.77/1529.4/58.85 | 0.401/0.654/16.993 |
| Python | Fib(10/20/30) | 633.68/209.89/2.9 | 1.578/4.764/345.307 |
| PHP | Fib(10/20/30) | 2051.22/168.8/1.78 | 0.488/5.942/560.553 |

According to the results at above, we make several tests with Node.js in order to find performance difference in various concurrent users as the calculation increases.

FIG.7 and FIG.8 show the results of “Calculate Fibonacci (10/20/30))” as the users grow from 10 to 1000.

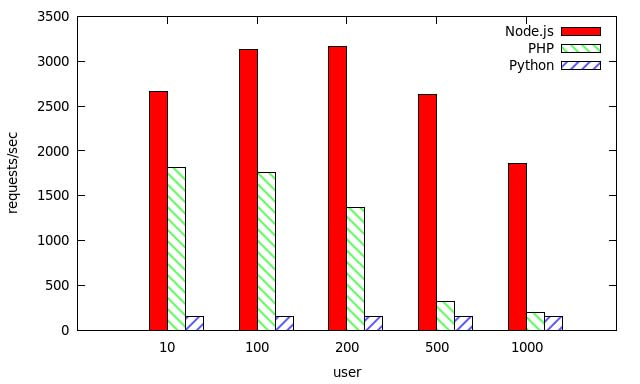


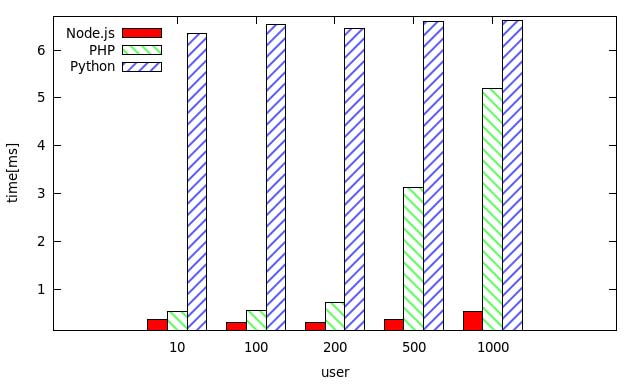


Although there is a little difference between the results of various concurrent users at the same calculation, the whole trend keeps stable. The “mean requests per second” of Fibonacci (10) is between 2000 and 2800. Fibonacci (20) is between 1300 and 2000. Fibonacci (30) is around 60. Meanwhile, the test of Fibonacci (30) is interrupted when users are up to 500. From the above, the increment of calculation brings much more effect than the larger users. We make a simple conclusion that Node.js is more adapted to IO-intensive application, not compute-intensive application, because compute-intensive applications don’t exploit good advantages of Node.js.

1. **“Select Operation of DB” module**

In order to validate the conclusion to prove Node.js is adapted to IO-intensive application, we design a “Select Operation of DB” module. FIG.9 and FIG.10 show the results of this module.





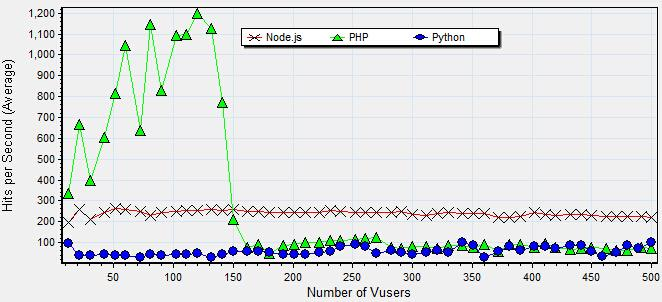
The max “mean requests per second” of Node.js is 3164.46, 20 times larger than Python-Web and 2 times larger than PHP. In addition, the peak value of “Select Operation of DB” module with Node.js doesn’t have much difference with “Hello World” module. The “mean requests per second” of Python- Web keeps around 150 and maintains stable as users increase. But to PHP, the “mean requests per second” slows down and the “mean time per request” is longer. It proves that Node.js is more suitable for IO-intensive application among the three, while PHP is applicable to small scale website.

1. **Results and Analyses of Scenario Tests**

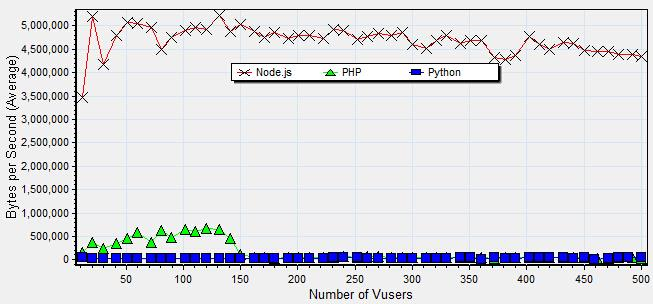
To validate the results on benchmark tests, we choose two scenarios as follows.

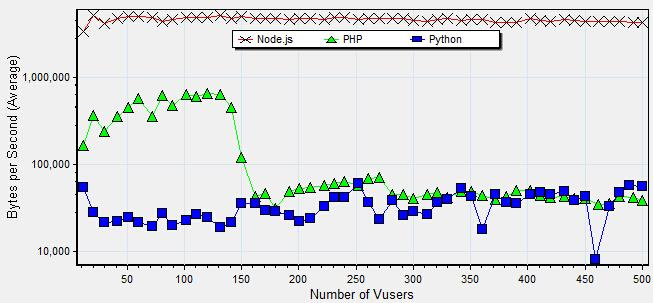
* + - 1. “Login” scenario

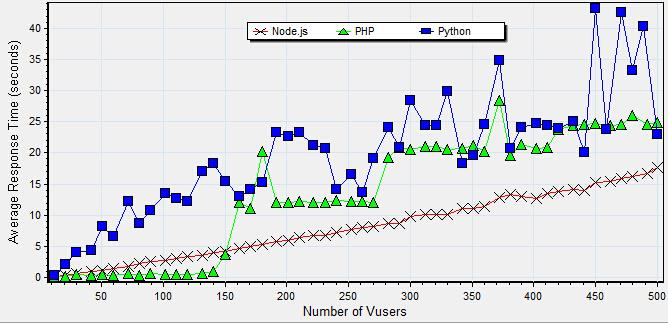
We choose peak users at 500 to do tests in “Login” scenario. We mainly observe “hits per second”, throughput and average transaction response time as users goes up. From FIG.11 to FIG.14, the results of “Login” scenario are shown. The horizontal axis represents the number of concurrent users, the vertical axis representing hits per second, throughput, throughput trend, average transaction response time. FIG.11 shows “hits per second” for different concurrent users. “Hits per second” measures the number of HTTP requests sent to Web server from virtual users per second in performance tests. “Hits per second” is larger and the stress to Web server is larger. It can be seen from FIG.11 that “hits per second” decreases in large degree when users are up to 150. It is to say the system can’t hold so many users.



In addition, throughput in FIG.13 slows down clearly when users are 150. Meanwhile, average transaction response time in FIG.14 increases much and it means the load of system is at a peak. It leads to longer response time with the increase of users, and it appears some users’ timeout. So “hits per second” in FIG.11 decreases. The throughput of Node.js is about 5,000,000 byte/s in FIG.12. However, the throughput of PHP and Python-Web are both below 1,000,000 byte/s. We can see a more detailed data from FIG.13 that the throughput of Node.js is between 4,000,000 and 5,000,000 byte/s. The throughput of PHP stays 500,000 byte/s when the users are less than 150 and it’s similar to Python-Web after that, keeping to 70,000 byte/s. In general, the throughput of Node.js is far more than PHP and Python, more adapted to IO-intensive requests. On the other hand, Node.js is more suited to concurrent situation, while PHP is applicable to middle and small-scale website. The average transaction response time of PHP and Node.js are very close within the first 150 users, even the time of PHP is less than Node.js in FIG.14. At the same time, the average transaction response time of PHP and Node.js are both less than 4s. After that, the response time of PHP is much larger than Node.js with users increasing. However, Node.js stays steady growth trend because its rate to deal with requests is higher than PHP with the concurrent users are up. Thus Node.js takes better place when user requests increase.

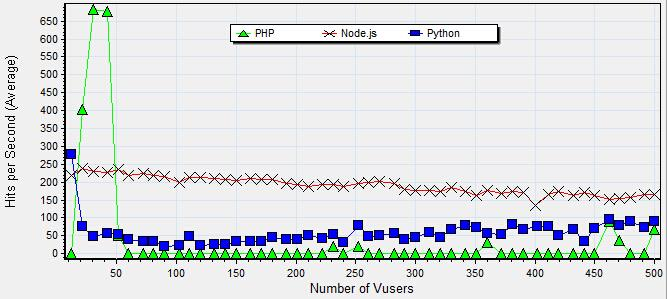


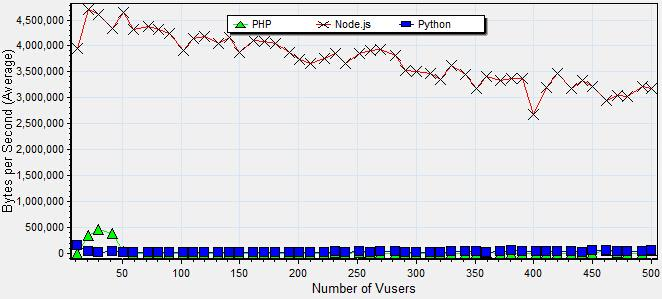


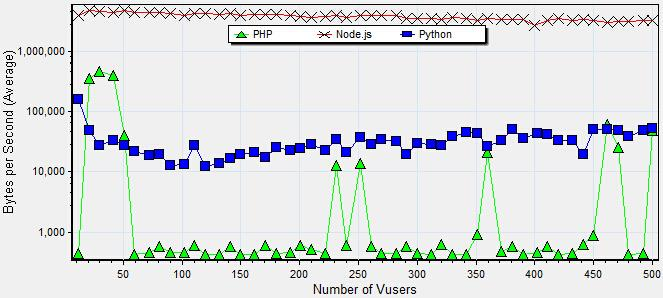


2) “Encryption” scenario

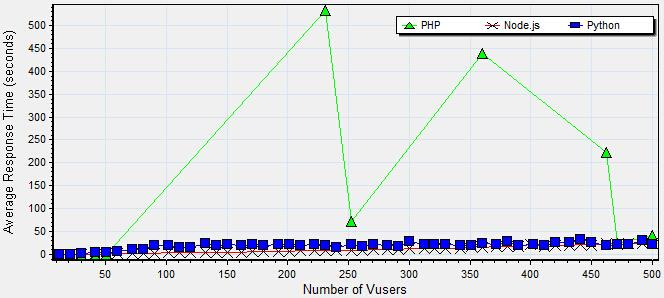
We simulate a real situation to encrypt password on basis of “Login” scenario in order to validate the performance of three technologies in compute-intensive situation. It also makes experiments when users are 500 at max. FIG.15 shows the results of “hits per second”. It can be seen from the figure that “hits per second” decreases in large degree when users are up to 50. The decrease time moves up compared to it in “Login” scenario because the new “encryption” scenario is more complex and reduces the performance of PHP. The stress server can undertake is to limit when the users are up to 50. It also can be seen in FIG.17 that the trend of throughput is relative to “hits per second” for PHP. 666 In FIG.16, the throughput of Node.js is 4,000,000 byte/s, going down 1,000,000byte/s contrasting with FIG.12. From the trend, the throughput of Node.js is keep between 3,000,000 byte/s and 4,000,000 byte/s. All at once, it’s steadily falling as users increase. PHP is similar to it in “Login” scenario when users are less than 50 and the throughput of PHP stays 500,000 byte/s now. Then it goes down by a large degree less than 1,000 byte/s. Python is alike with its performance of “Login” scenario to keep stable, but has a slight decline around 50,000 byte/s. In short, three technologies all aren’t adapted to compute-intensive application, especially PHP. However, “Encryption” scenario brings PHP the least effect among the three. Node.js is more suitable for IO-intensive application rather than compute-intensive sites.







The average transaction response time is shown in FIG.18 and the time of PHP shows very unstable with the increasing users. On one hand, it’s due to the effect of “Encryption” scenario. On the other hand, it’s the mode of multi-process in PHP. Nevertheless, the response time of PHP and Node.js are in a good slowly increasing trend with the increasing users.



Related Work

There have been lots of studies evaluating and analysing Web server performance. Lance and Martin experimentally evaluated the impact of three different dynamic content technologies (Perl, PHP, and Java) on Web server performance [4]. The results showed that the overheads of dynamic content generation can reduce the peak request rate supported by a Web server up to a factor of 8. Meanwhile, the results indicated that Java server technologies typically outperform both Perl and PHP for dynamic content generation. Scott used the SPECWeb2005 benchmark to contrast the performance of PHP and JSP with the popular Web servers Apache and Lighttpd according to the enterprise standard and made a conclusion that JSP performs better than PHP [5]. Alok compared the performance of ASP.NET, JSP and PHP using 4 benchmarks [6]. J. Hu and Y. Hu evaluated Web Server performance in LAN environments, but they only concerned static Web content [7, 8]. E. Cecchet used two distinct benchmarks, including online bookstore and auction sites, to identify more general performance trade-offs relevant to any site using dynamic Web content generation [9]. Ramana analysed the performance differences between PHP and compiled languages such as C Language, pointing out the relative performance downside of PHP [10]. Warner described the importance to use Web technology such as PHP rather than just JSP for real-world benchmarking [11]. Pedro compared from the two aspects of Java and PHP in performance and security, and then made a conclusion that PHP is more scalability and security relative to Java [12]. The research as above focused on the comparison of Web development technologies. But to the best of our knowledge, all of these studies don’t consider the newest Web technology, such as Node.js. The paper is the first one to compare the performance of those three popular Web technologies, Node.js, PHP and Python-Web.

Bibliography

[1] http://www.theserverside.com/news/1363671/What-is-an-App-Server

[1] http://Node.js.org

[2] http://strongloop.com/developers/Node-js-infographic/?utm\_source=ourjs.com#3

[3] https://github.com/search?l=JavaScript&o=desc&q=stars%3A%3E1&s=stars&type=Repositories.

[4] T.Lance, A.Martin and W.Carey, “Performance Comparison of Dynamic Web Technologies”, ACM SIGMETRICS Performance Evaluation Review, Volume 31 Issue 3, December 2003.

[5] T.Scott, T.Michiaki, S.Toyotaro, T.Akihiko, and O.Tamiya, “Performance Comparison of PHP and JSP as Server-Side Scripting Languages”, Middleware, 2008.

[6] A.Ranjan, R.Kumar, J.Dhar, “A Comparative Study between Dynamic Web Scripting Languages”, Data Engineering and Management, 2012.

[7] J.Hu, S.Mungee, and D.Schmidt, “Techniques for Developing and Measuring High-Performance Web Servers over ATM Networks", Proceedings of IEEE INFOCOM, San Francisco, CA, March/April 1998.

[8] Y.Hu, A.Nanda, and Q.Yang, “Measurement, Analysis, and Performance Improvement of the Apache Web Server", Technical Report No. 1097-0001, University of Rhode Island, 1997.

[9] E.Cecchet, A.Chanda, S.Elnikety, J.Marguerite, and W.Zwaenepoel, “Performance Comparison of Middleware Architectures for Generating Dynamic Web Content", Proceedings of 4th Middleware Conference, Rio de Janeiro, Brazil, June 2003.

[10] U.Ramana, T.Prabhakar, “Some Experiments with the Performance of LAMP Architecture”, Proceedings of the 2005 Fifth International Conference on Computer and Information Technology, 2005.

[11] S.Warner, J.Worley, “SPECWeb2005 in the Real World: Using Internet Information Server (IIS) and PHP”, 2008 SPEC Benchmark Workshop, 2008.

[12] P.Neves, N.Paiva, J.Durães, “A comparison between JAVA and PHP”, C3S2E '13 Proceedings of the International C\* Conference on Computer Science and Software Engineering, 2013.

[13] http://www.PHP.net/.

[14]

[15] R.Jain, “The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling”, John Wiley & Sons, Inc., New York, NY, 1991.

[16] S.Tilkov, S.Vinoski, “Node.js: Using JavaScript to Build High- Performance Network Programs”, IEEE Internet Computing, 2010.